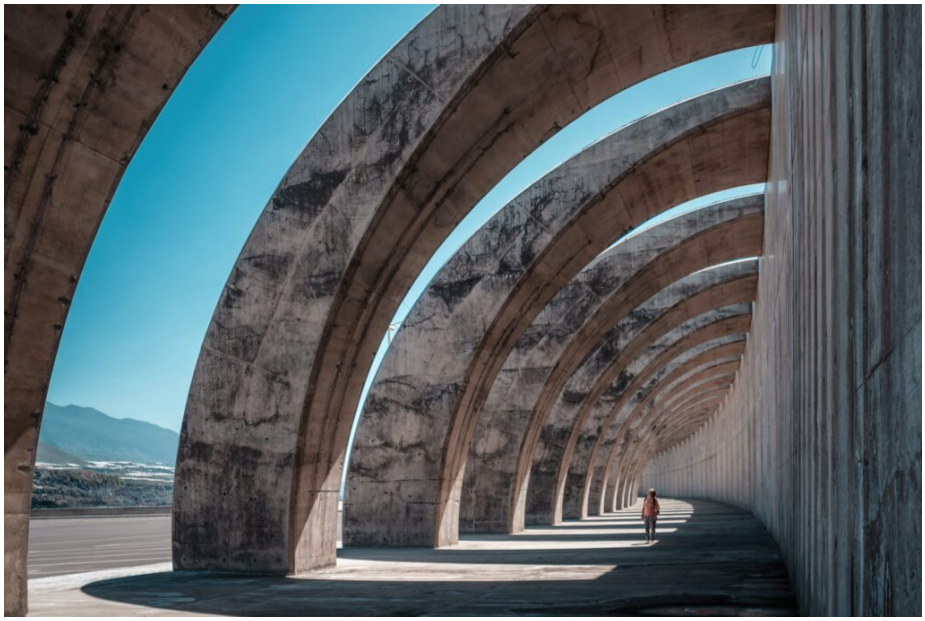
Issue 065: Pascal

Adrian Kosmaczewski

February 5th, 2024



Welcome to the sixty-fifth issue of *De Programmatica Ipsum*, about the *Pascal programming language.*

In this edition:

- We react to the sudden news of Niklaus Wirth's passing[1] with memories of the past and perspectives of the present.
- In the Library section[2], we review "Classics in Software Engineering" by Edward Nash Yourdon[3].
- In our Vidéothèque section[4], we watch some recent interviews of Niklaus Wirth[5] himself.

We would like to thank our patrons who generously contribute every month (or have contributed in the past) to our work and help us run this magazine. Thank you so much! In alphabetical order: Adam Guest, Adrian Tineo Cabello, Benjamin Sheldon, Christopher Nascone, Franz Lucien Moersdorf, Guillermo Ramos Álvarez, Jean-Paul de Vooght, Patryk Matuszewski, Paul Hudson, Quico Moya, Roger Turner, and Szymon Licau.

Enjoy this issue! Please subscribe to our free newsletter[6] to stay updated about new releases, share the articles on social media, or contribute[7] if you would like to support our work with

[1] https://deprogrammaticaipsum.com/lazarus-come-forth/
[2] https://deprogrammaticaipsum.com/category/library/
[3] https://deprogrammaticaipsum.com/edward-nash-yourdon/
[4] https://deprogrammaticaipsum.com/category/videotheque/
[5] https://deprogrammaticaipsum.com/niklaus-wirth/
[6] https://deprogrammaticaipsum.com/newsletter/
[7] https://deprogrammaticaipsum.com/contribute/

a donation via Liberapay[8].

Cover photo by Fausto García-Menéndez[9] on Unsplash[10].

[8] https://liberapay.com/

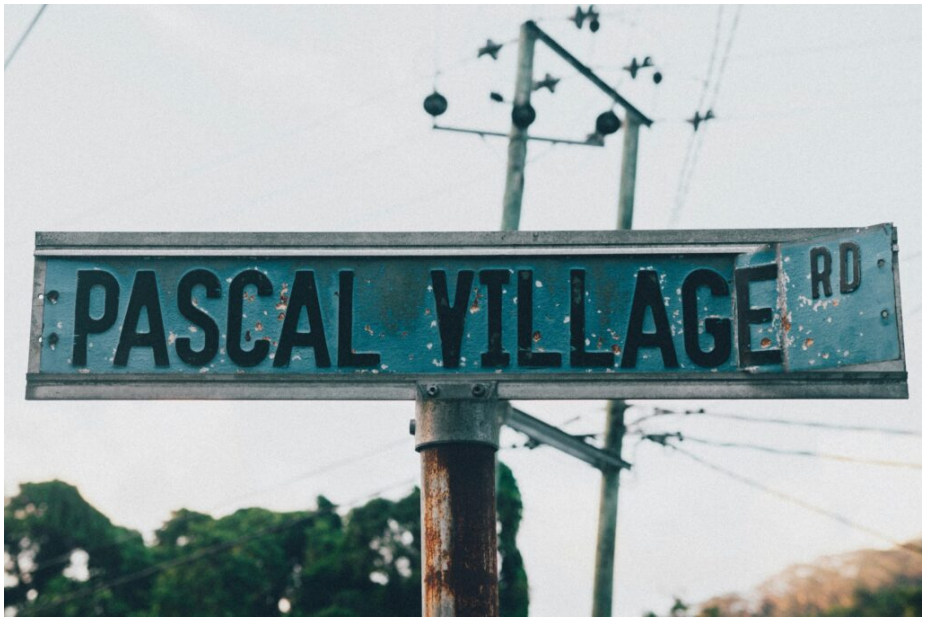
[9] https://unsplash.com/@faustogarmen?utm_content=creditCopyText&utm_medium=referral&utm_source=unsplash

[10] https://unsplash.com/photos/person-standing-on-hallway-taken-during-daytime-ChqVNNr8dF4?utm_content=creditCopyText&utm_medium=referral&utm_source=unsplash

“Lazarus, Come Forth!”

Adrian Kosmaczewski

February 5th, 2024



The year 2024 started with the sudden and sad news[1] of the passing away of Niklaus Wirth, Turing Award winner[2] and creator of many influential programming languages. It was hard for this author not to dive back into memories of Pascal, probably Wirth’s more successful and famous creation, for the main article of this month. This will not be an obituary; after all, The Register published one that, I believe, is the perfect one[3]. Instead, we will focus on the myriad breadcrumbs of evidence showcasing the towering legacy of Mr. Wirth.

Younger generations of programmers, prone to `npm install` whichever new library makes the rounds, cannot sadly fully realize the kind of power that the Pascal programming language put in the hands of personal computer users in the 1980s. This power was not just limited to the IBM PC, but also on the Apple Mac, whose first official development environment[4] was, indeed, based on another dialect of Pascal.

If BASIC[5] was the language of the 1970s, and Java[6] the language of the 1990s, Pascal was undoubtedly the language of the 1980s.

I got my first copy of Turbo Pascal around 1992, a few months after I bought my first PC. After hitting the limits of what QBasic could offer, my late friend Bertrand Dufresne highly recommended Borland Turbo Pascal to me, and needless to say, it was a one-way trip. Turbo Pascal had some incredible features, some of which still stand (and sometimes surpass[7]) the

[1] https://lists.inf.ethz.ch/pipermail/oberon/2024/016856.html
[2] https://www.youtube.com/watch?v=SUgrS_KbSI8
[3] https://www.theregister.com/2024/01/04/niklaus_wirth_obituary/
[4] https://deprogrammaticaipsum.com/eternally-finally/
[5] https://deprogrammaticaipsum.com/programming-the-liberal-arts/
[6] https://deprogrammaticaipsum.com/java-the-programmer-environment-that-has-it-all/
[7] https://blogsystem5.substack.com/p/the-ides-we-had-30-years-ago-and

test of our 21st century demands. It featured a full-screen IDE, with an integrated editor, debugger, variable watcher, multiple file edition, and a now iconic and very pleasing blue background color, today still installed by default on Vim (just one `:color blue` command away).

Turbo Pascal generated self-contained, small, and efficient `.EXE` programs for MS-DOS. Just hit `F9` and save your final product on a floppy disk, share it with your friends, or even better, with the wider world, as a shareware[8], for example. Many a programming career started following the workflow I just described, thanks to a tool designed and implemented by a Danish programmer named Anders Hejlsberg[9]. Remember this name.

My interactions with Borland IDEs and the Pascal programming language dominated my first five years of programming experience. In 1993, as I started studying physics, we had a two-semester course in computer programming, and lo and behold, the professor chose Turbo Pascal as a tool for learning. Sadly, his lack of pedagogy skills took him to consider the option of making his students write code during examinations on paper, without a computer, and that missing semicolons were worth half a mark. Many of my peers lost all interest in programming during that year. Well done, professor whatever-your-name-was.

In 1995, another friend suggested I give a shot to Borland Delphi[10], a new IDE for Windows 3.1 and Windows 95. Delphi was taking more than a few cues from Microsoft Visual Basic, but with a far more powerful language built-in. This last phrase is an oxymoron; Delphi made Visual Basic look like a botched experiment similar to a genetic cross between Donald Duck and the Chernobyl nuclear plant.

Of course, 1995 was also the year Java was released. I do not think Borland realized how much their world was going to change from that point onwards. At some point, they decided that renaming the company to "Embarcadero" was a good idea. As a result, their brand vanished almost overnight; by the time of the Dot-Com Crash, they were gone from the collective psyche of a whole generation of programmers. Never underestimate the power of a good brand; Borland enjoyed a solid reputation, even better than they even suspected.

One of the major casualties of the implosion of Borland was Anders Hejlsberg himself, who promptly found refuge in Microsoft to continue his already illustrious career. If the name of Visual J++ does not ring a bell, I am pretty sure that C# and TypeScript do ring more than a few. Well, he is the mastermind behind those three languages; bragging rights guaranteed. If the Turing Award is the Oscars' equivalent for computer programming, Hejlsberg deserves at least the People's Choice Award. At least. His contributions to developer experience are unparalleled.

Thankfully, the Open-Source movement did not give up on Pascal and promptly started to produce an alternative. 25 years later, I decided to install Lazarus[11], the most advanced Pascal-based IDE available today, on my personal computer running Fedora 39. In a sad twist of destiny, I also learned that Lazarus 3.0 was released[12] merely 10 days before Niklaus Wirth passed away.

(As a personal note, let it be known that I have always found the religious[13] undertones of the name "Lazarus" to be of a certain bad taste. I think this product deserved a better name.

[8] https://en.wikipedia.org/wiki/Shareware
[9] https://en.wikipedia.org/wiki/Anders_Hejlsberg
[10] https://en.wikipedia.org/wiki/Delphi_(software)
[11] https://www.lazarus-ide.org/
[12] https://forum.lazarus.freepascal.org/index.php/topic,65612.0.html
[13] https://en.wikipedia.org/wiki/Lazarus_of_Bethany

But naming is hard, yadda yadda. To add insult to injury, when Jesus ordered Lazarus of Bethany to wake up from among the dead, he did it by summoning a different programming language[14]. There is a strong chance that Jesus did not speak English, though. I give you that.)

Just two downloads and a quick `sudo dnf install` command later, and I launched the IDE. The "about box" shown during the startup of Lazarus features an interesting phrase: "Write Once, Compile Anywhere." Rings a bell.[15] Of course, I was at first put off by the multiple windows spread all over my screen, but a quick search brought the solution[16] to my eyes.

After this first hiccup, I created a project and was promptly dragging and dropping controls on a form (in this case, a button and a label.) I double-clicked on the button to add some code (more precisely to change the text of the label), and the `F9` key (sounds familiar?) brought the ensemble to life in less than a second.

"I double-clicked on the button", I just said. If this does not remind you of Visual Basic and Delphi 1.0, nothing will. Ah, memories.

And the code? Well, to put it in somewhat colorful terms, definitely unworthy of its legacy and impact, Pascal is a well-structured and... yes, uptight programming language. Everything has its place, an expected behavior, and a reason to be. In a sense, yes, Pascal is very Swiss in its essence.

The cultural roots of Mr. Wirth, born and raised in the Swiss city of Winterthur[17], are visible throughout the syntax and the semantics of each token. Pascal programs begin with the... `program` keyword, and are correctly terminated with a period at the bottom. All variables are neatly (and strongly[18]) defined before any code using them. Procedures are procedures and functions are functions. Couples of `begin` and `end` statements clearly delimit blocks of code. This is not some sloppy "west-coast" programming language, as I once described[19] Objective-C or Ruby, but the uptight European cousin of "east-coast" programming languages such as C and C++.

I like to think of programming languages in those terms. In Pascal, there is an order to things because code was meant to be read by humans more than it was meant to be compiled by computers.

Later versions of Turbo Pascal already featured Object-Oriented[20] characteristics close to those of C++; heap allocation (through `NEW` and `DISPOSE` keywords); and even pointers–in the case of Pascal, denoted using a caret ("^") character next to the type of the variable, instead of the asterisk ("*") used in C or C++. Alas, the wide community of programmers wanted garbage-collected runtimes in the 1990s, not pointers and manual memory management; and thus the fate of Pascal in general was sealed in the marketplace.

The Lazarus IDE is cross-platform[21], available for Windows, Mac, and Linux, in both 32 and 64 bits (although I do not know how good is the support for more recent chipsets with ARM architectures, if any.) Not many commercial systems nowadays can brag about the same level of availability and features for such a low price point. And for real Turbo Pascal nostalgia, try

[14] https://en.wikipedia.org/wiki/Forth_(programming_language)
[15] https://deprogrammaticaipsum.com/write-anywhere-run-once/
[16] https://www.pragmaticlinux.com/2022/07/single-window-mode-for-the-lazarus-ide/
[17] https://en.wikipedia.org/wiki/Winterthur
[18] https://deprogrammaticaipsum.com/the-truce-of-type-inference/
[19] https://akos.ma/blog/the-developer-guide-to-migrate-across-galaxies/
[20] https://deprogrammaticaipsum.com/the-hype-cycle-of-oop/
[21] https://deprogrammaticaipsum.com/dr-dobbs-and-the-deathly-cross-platform-app/

using the `fp` command and launching the text mode IDE with its glorious blue background color. You will thank me later.

I cannot avoid pouring a tear thinking that my friend Bertrand and Mr. Wirth are somewhere up there talking about Pascal and other programming languages, wherever that might be, however that might work.

Cover photo by Dario Didon[22] on Unsplash[23].

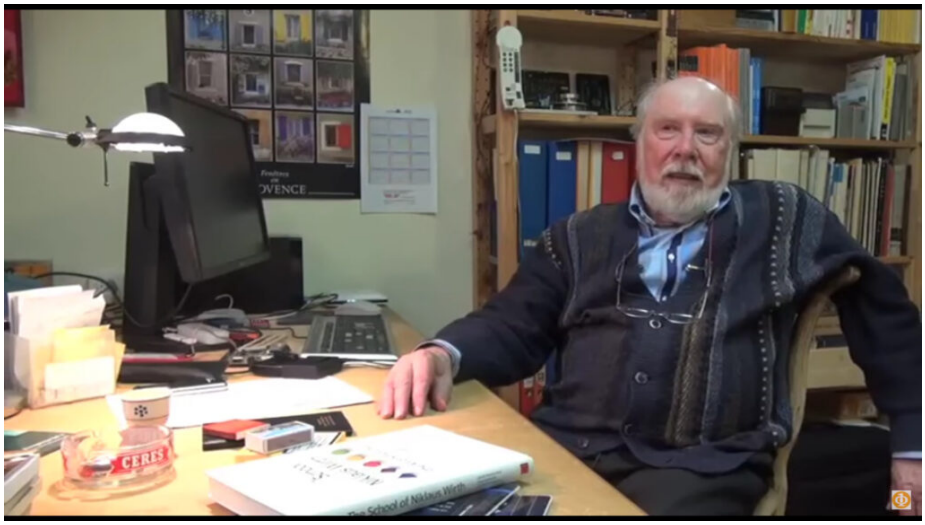
[22]https://unsplash.com/@d_dmd2?utm_content=creditCopyText&utm_medium=referral&utm_source=unsplash

[23]https://unsplash.com/photos/a-street-sign-that-reads-pascal-village-rd-ViQAOgLk0o0?utm_content=creditCopyText&utm_medium=referral&utm_source=unsplash

Niklaus Wirth

Adrian Kosmaczewski

February 5th, 2024



It is challenging to summarize the influence of Niklaus Wirth[1] in the daily lives of otherwise unsuspecting programmers worldwide. The more one digs into videos, papers, books, and obituaries, the more information surfaces and fights for a place in the spotlight. There are, however, at least two major guidelines that drove his passion for software. One was simplicity through clear understanding and lack of ambiguity; the other, closely related to the first, was teaching.

This month's Vidéothèque video is an interview of Mr. Wirth[2] published by the IEEE Computer Society. It is a short but nevertheless revealing video, where those two guiding lights appear clearly in each of the anecdotes.

Wirth's aim for simplicity was not entirely different from, say, Yukihiro Matsumoto[3]'s wishes for Ruby. Quoting its creator, Ruby was built to (and I quote) "make programmers productive and happy". The question is, what is happiness? For Mr. Wirth the answer was simple, and it consisted of clear understanding, learning, and progress.

In his conference paper "On the Design of Programming Languages"[4] for the IFIP Congress of 1974, Mr. Wirth states this point with clarity:

> The key, then, lies not so much in minimising the number of basic features of a language, but rather in keeping the included facilities simple to understand in all their consequences of usage and free from unexpected interactions when they are combined. (...) The language should not be *burdened* with syntactical rules, it must be *supported* by them. They must therefore be purposeful, and prohibit the constructions of ambiguities.

[1] https://en.wikipedia.org/wiki/Niklaus_Wirth
[2] https://www.youtube.com/watch?v=BJIqHIYSDrk
[3] https://en.wikipedia.org/wiki/Yukihiro_Matsumoto
[4] https://www.semanticscholar.org/paper/On-the-Design-of-Programming-Languages-Wirth/9e3f2c3e559ff8df19fc7900fa7453a23cb36c34

Looking even further back in time, as far as the year 1965[5], helps us find more references to this quest of simplicity:

> The result of a systematic language design based on consistent rules is reliability on the part of the implemented translator as well as on the part of the user, who is discouraged from using "tricks" and is less subject to pitfalls and misunderstandings about the nature of complicated or ill-defined facilities.

Let us look at the list of programming languages created by Niklaus Wirth (and try not to gasp): Euler[6] in 1965, PL360[7] in 1966, ALGOL W[8] in 1966, Pascal[9] in 1970, Modula[10] in 1975, Modula-2[11] in 1978, Oberon[12] in 1987, Oberon-2[13] in 1991, and Oberon-07[14] in 2007. For context, and to our amazement, the first and the last languages in the list span a period of 42 years.

Three major names stand out from the list above: Pascal, Modula, and Oberon. Notwithstanding the obvious similarities in syntax between the three (a fact somewhat acknoledged by Wirth in the video), they represented gradual steps in the same direction, towards the creation of simpler, smaller, faster software. The pinnacle of this work was Oberon, a project that yielded two major contributions simultaneously: a compact and fast operating system[15], and the programming languages enumerated in the previous paragraph.

In the 1990s, as soon as Internet connectivity became ubiquitous, I downloaded a copy of Oberon from the FTP server of the Swiss Federal Institute of Technology Zürich (ETHZ) and ran it on my personal computer. It was an unwieldy but fascinating system that booted in nanoseconds, required very little in terms of resources, and even provided a complete, albeit rudimentary, mouse-powered graphical user interface. I recently found a video featuring a review of Oberon[16] that I can recommend everyone to watch.

The achievement of Oberon was most excellently described in his 1995 article "A Plea for Lean Software"[17] published in IEEE Computer magazine:

> The belief that complex systems require armies of designers and programmers is wrong. A system that is not understood in its entirety, or at least to a significant degree of detail by a single individual, should probably not be built.

Let us ponder on these words, as we face 2024 with a still unresolved debate about how to explain the internal mechanisms of Large Language Models[18], nowadays powering the new Generative AI systems that are making the headlines every day since November 2022.

Many of the creators of those LLMs and other modern systems have probably never read Niklaus Wirth's 10-page paper[19] published in the IEEE Annals of the History of Computing

[5] https://dl.acm.org/doi/10.1145/321439.321442
[6] https://en.wikipedia.org/wiki/Euler_(programming_language)
[7] https://en.wikipedia.org/wiki/PL360
[8] https://en.wikipedia.org/wiki/ALGOL_W
[9] https://en.wikipedia.org/wiki/Pascal_(programming_language)
[10] https://en.wikipedia.org/wiki/Modula
[11] https://en.wikipedia.org/wiki/Modula-2
[12] https://en.wikipedia.org/wiki/Oberon_(programming_language)
[13] https://en.wikipedia.org/wiki/Oberon-2
[14] https://en.wikipedia.org/wiki/Oberon_(programming_language)#Oberon-07
[15] https://en.wikipedia.org/wiki/Oberon_(operating_system)
[16] https://www.youtube.com/watch?v=OJGnpmnXR5w
[17] https://ieeexplore.ieee.org/document/348001
[18] https://arxiv.org/abs/2309.01029
[19] http://ieeexplore.ieee.org/document/4617912/

in July 2008. On that occasion, Niklaus Wirth asserted an everlasting principle, one that has guided this magazine since its inception:

> It is unfortunate that people dealing with computers often have little interest in the history of their subject. As a result, many concepts and ideas are propagated and advertised as being new, which existed decades ago, perhaps under a different terminology.

As Liam Proven says in the closing words of his excellent obituary of Mr. Wirth[20],

> The modern software industry has signally failed to learn from him. Although he has left us, his work still has much more to teach.

Watch this month's Vidéothèque video of Niklaus Wirth, 1984 Turing Award winner[21], on the YouTube channel of the IEEE Computer Society[22]. But do not stop there. There is, fortunately, lots of material by Mr. Wirth available online–and quite recent, and of excellent quality, too.

He stayed active long after retirement, publishing papers and working with his peers of the ETHZ. We can now peruse hours of material, and aim to become better software engineers during the process. Suffice to mention this long 2018 interview[23] by the Association of Computing Machinery (of which there is a short excerpt[24] specifically about Pascal); the recording of a 2014 symposium[25] at the ETHZ; an interview also conducted in German (subtitles are available) in 2021 at ETHZ by Friedemann Mattern[26], available in three[27] short[28] fragments[29]; and another 2021 interview[30] by Michael Holzheu (also in German).

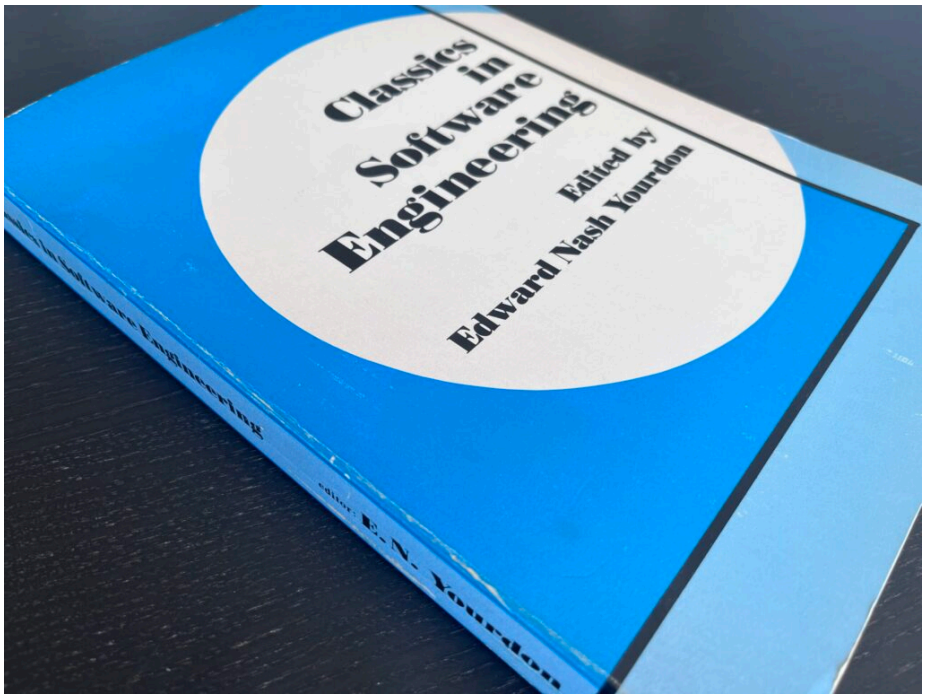
Cover snapshot chosen by the author from the video.

[20] https://www.theregister.com/2024/01/04/niklaus_wirth_obituary/
[21] https://amturing.acm.org/award_winners/wirth_1025774.cfm
[22] https://www.youtube.com/watch?v=BJIqHIYSDrk
[23] https://www.youtube.com/watch?v=SUgrS_KbSI8
[24] https://www.youtube.com/watch?v=F7uZsSbodHw
[25] https://www.youtube.com/watch?v=EXY78gPMvl0
[26] https://en.wikipedia.org/wiki/Friedemann_Mattern
[27] https://www.youtube.com/watch?v=eAUS02ec3ow
[28] https://www.youtube.com/watch?v=KQvmo89lPSE
[29] https://www.youtube.com/watch?v=DWyrAv1QY7c
[30] https://www.youtube.com/watch?v=OEmMx55SF8U

Edward Nash Yourdon

Adrian Kosmaczewski

February 5th, 2024



Here is a confession. The first drafts of this issue of De Programmatica Ipsum were written under the name *"Structured Programming"*. Understandably enough, the news[1] of Niklaus Wirth's passing triggered a prompt renaming and the choice of a somewhat narrower focus. However, Pascal's rise in popularity during the 1970s and 1980s cannot be explained unless we elaborate on Structured Programming, and this month's Library book is among the most important ones ever written about the subject.

The phrase "Structured Programming" implies the existence of an unstructured kind thereof. We have seen some examples of Unstructured Programming during the discussion of BASIC[2] a few months ago. The keyword that shows the lack of structure is `GOTO`, of course. Pascal showed up in history as the archnemesis of BASIC, and kept this role until the rise of Object-Oriented Programming[3] as the dominant paradigm in the 1990s, until the Delphi vs. Visual Basic debate was promptly drowned by the rise of Java and C#.

Let us analyze the genesis of Pascal, which occurred thanks to three major events that took place in the same *annus mirabilis* 1968.

Act one: in March, Edsger Dijkstra publishes his (in)famous "Go To Statement Considered Harmful"[4] article in the March 1968 edition of Communications of the ACM, probably the most cited paper ever in the history of computer programming. (This paper is, by the way,

[1] https://lists.inf.ethz.ch/pipermail/oberon/2024/016856.html
[2] https://deprogrammaticaipsum.com/programming-the-liberal-arts/
[3] https://deprogrammaticaipsum.com/the-hype-cycle-of-oop/
[4] https://homepages.cwi.nl/~storm/teaching/reader/Dijkstra68.pdf

reprinted on page 29 of "Classics of Software Engineering", this month's Library book. But let us not digress.)

Act two: in October takes place the first, now legendary, NATO Software Engineering Conference[5] where the words "software crisis" fueled passionate discussions[6]:

> There was a considerable amount of debate on what some members chose to call the "software crisis" or the "software gap". As will be seen from the quotations below, the conference members had widely differing views on the seriousness, or otherwise, of the situation, and on the extent of the problem areas.

As the astute reader can imagine, the lack of structure was one of the major drivers of this "crisis".

Act three: in December, the ALGOL committee (wrongly) chooses Adriaan van Wijngaarden[7]'s ideas as the basis for ALGOL 68[8], instead of Niklaus Wirth's much simpler proposal, historically referred to as ALGOL W[9].

Niklaus Wirth promptly thanked the courtesy and left the ALGOL commitee, taking with him the ideas of ALGOL W. He framed them with the family name of a famous philosopher of the seventeenth century[10], thus creating a programming language whose name[11] conveys seriousness, logic, and most importantly, clarity of thought[12].

The first Pascal compiler (single-pass, top-down, recursive-descent based) appeared around 1970, after a failed first attempt in 1969, and the language slowly grew in popularity among teachers of computer science curricula.

Wirth told the story of Pascal in the second History of Programming Languages conference[13] of 1993:

> The primary innovation of Pascal was to incorporate a variety of data types and data structures, similar to ALGOL's introduction of a variety of statement structures. ALGOL offered only three basic data types: integers, real numbers, and truth values, and the array structure; Pascal introduced additional basic types and the possibility to define new basic types (enumerations, subranges), as well as new forms of structuring: record, set, and file (sequence), several of which had been present in COBOL.

The major historical breakthrough of the language was, of course, the UCSD Pascal[14] system of the University of California San Diego in 1977, at the basis of various commercial implementations, including Apple's own version[15] for the Apple II.

"Structured Programming" was the big paradigm of the 1970s and 1980s, well before "Object Orientation" entered the collective psyche, and Pascal was poised to be its most important

[5] https://en.wikipedia.org/wiki/NATO_Software_Engineering_Conferences
[6] http://homepages.cs.ncl.ac.uk/brian.randell/NATO/nato1968.PDF
[7] https://en.wikipedia.org/wiki/Adriaan_van_Wijngaarden
[8] https://en.wikipedia.org/wiki/ALGOL_68
[9] https://en.wikipedia.org/wiki/ALGOL_W
[10] https://en.wikipedia.org/wiki/Blaise_Pascal
[11] https://en.wikipedia.org/wiki/Pascal_(programming_language)
[12] https://en.wikipedia.org/wiki/Pens%C3%A9es
[13] https://en.wikipedia.org/wiki/History_of_Programming_Languages_(conference)
[14] https://en.wikipedia.org/wiki/UCSD_Pascal
[15] https://en.wikipedia.org/wiki/Apple_Pascal

messenger. The `GOTO` keyword was to be erased from the minds of software developers forever, following Dijkstra's plea for sanity.

Yet... in an interesting twist of pragmatism, the first versions of Pascal *did include it*. Yes, you heard right; it was there, as a way to help software developers migrate their code from other languages (read, BASIC and FORTRAN) into this new world of procedures and functions. Again, let us read Niklaus Wirth's own words, telling us the reason in his HOPL II paper:

> The most vociferous criticism (of Pascal) came from Habermann, who correctly pointed out that Pascal was not the last word on language design. Apart from taking issue with types and structures being merged into a single concept, (...) he reproached Pascal for retaining the much-cursed **goto** statement. In hindsight, one cannot but agree; at the time, its absence would have deterred too many people from trying to use Pascal. The bold step of proposing a **goto-less** language was taken ten years later by Pascal's successsor Modula-2.

Let us talk now about the actual book of this month: "Classics in Software Engineering", a recollection of papers compiled and published in 1979 by Edward Nash Yourdon[16], a computer consultant and engineer well known for his work in notations used in structured and object-oriented programming.

The `GOTO` keyword plays a central role in the book, almost to the point of obsession; apart from Dijkstra's paper previously mentioned in this page, there literally are four more articles with the phrase "go to" in its title:

1. One in favor[17], by Martin Hopkins[18] of IBM, one of the inventors of the RISC architecture.
2. One against[19], by William Wulf[20].
3. A third one explaining how to translate "go to" programs to "while" programs[21], by Edward Ashcroft[22] and Zohar Manna[23].
4. Finally, Donald Knuth's own "Structured Programming with go to Statements"[24] paper.

"Classics in Software Engineering" proudly wears its name, featuring many more lectures and papers on the subject of Structured Programming by quite a few luminaries we have talked about in past issues of this magazine: Brian Kernighan[25], Barry Boehm[26], Tom DeMarco[27], and Donald Knuth[28]. To top it all, it contains a copy of another classic article by Edsger Dijkstra: his own Turing Award[29] lecture, "The Humble Programmer"[30], a timeless and enlightening lecture (or audio[31], if you prefer) recommended to all software engineers.

[16] https://en.wikipedia.org/wiki/Edward_Yourdon
[17] https://www.cs.csustan.edu/~mmartin/teaching/CS4100F19/Lectures/HopkinsGOTO.pdf
[18] https://www.legacy.com/us/obituaries/lohud/name/martin-hopkins-obituary?id=16857996
[19] https://dl.acm.org/doi/10.1145/800194.805861
[20] https://en.wikipedia.org/wiki/William_Wulf
[21] https://dl.acm.org/doi/abs/10.5555/1241515.1241521
[22] https://dl.acm.org/profile/81100071527
[23] https://engineering.stanford.edu/news/computer-science-pioneer-zohar-manna-dies-age-79
[24] https://dl.acm.org/doi/10.1145/356635.356640
[25] https://deprogrammaticaipsum.com/brian-kernighan/
[26] https://deprogrammaticaipsum.com/barry-boehm/
[27] https://deprogrammaticaipsum.com/tom-demarco-timothy-lister/
[28] https://deprogrammaticaipsum.com/the-art-of-the-art-of-computer-programming/
[29] https://amturing.acm.org/award_winners/dijkstra_1053701.cfm
[30] https://dl.acm.org/doi/pdf/10.1145/355604.361591
[31] https://www.youtube.com/watch?v=9hWv2ECXHng

And of course, the book features a contribution by Niklaus Wirth himself: "On the Composition of Well-Structured Programs"[32], introduced by Yourdon on page 151 as follows:

> For example, after four pages of a delightful philosophical review of the dismal state of the art of programming, Wirth launches into examining a total of five sample programs – all involving an ALGOL-like pseudocode that will be somewhat unfamiliar to the average COBOL programmer, and all involving applications that he wouldn't care about. If you're a COBOL programmer, all I can say is, be patient! Spend the time to read through the examples and to study the code that Wirth presents; it really is worth the effort.

I do not have much more to say, so I will just paraphrase and relay the same message to the audience of 2024: *if you are a JavaScript or Rust programmer, all I can say is, be patient! Spend the time to read this book; it really is worth the effort.*

If you want to read more from Niklaus Wirth himself, I can only recommend his classics, some of which are freely available online: his bestseller "Algorithms + Data Structures = Programs"[33] (1976); "Project Oberon: The Design of an Operating System, a Compiler, and a Computer"[34] (1992, revised in 2005 and 2013); and his lesser-known "Compiler Construction"[35] (1996). OK, here is one more, albeit not by Wirth, but about him: "The School of Niklaus Wirth"[36] (2000).

A final observation before closing this article. As much as I love (and have frequently mentioned in the pages of this magazine) Biancuzzi & Warden's wonderful book "Masterminds of Programming"[37], I cannot but feel angry (dare I say, "betrayed"?) that it does not include an interview of Niklaus Wirth, who was evidently alive and very much active in 2009. This is, however, the only complaint that I have of an otherwise wonderful book that will have its own entry in this section in the future, and I want to think that the reasons for this omission were none other than scheduling conflicts or commercial deadlines.

Cover photo by the author.

[32] https://dl.acm.org/doi/10.1145/356635.356639
[33] https://en.wikipedia.org/wiki/Algorithms_%2B_Data_Structures_%3D_Programs
[34] https://people.inf.ethz.ch/wirth/ProjectOberon/PO.System.pdf
[35] https://people.inf.ethz.ch/wirth/CompilerConstruction/CompilerConstruction1.pdf
[36] https://shop.elsevier.com/books/the-school-of-niklaus-wirth/boszormenyi/978-0-08-057418-9
[37] https://www.oreilly.com/library/view/masterminds-of-programming/9780596801670/